Natural Language to Visualization by Neural Machine Translation

Yuyu Luo, Nan Tang, Guoliang Li*, Jiawei Tang, Chengliang Chai, Xuedi Qin

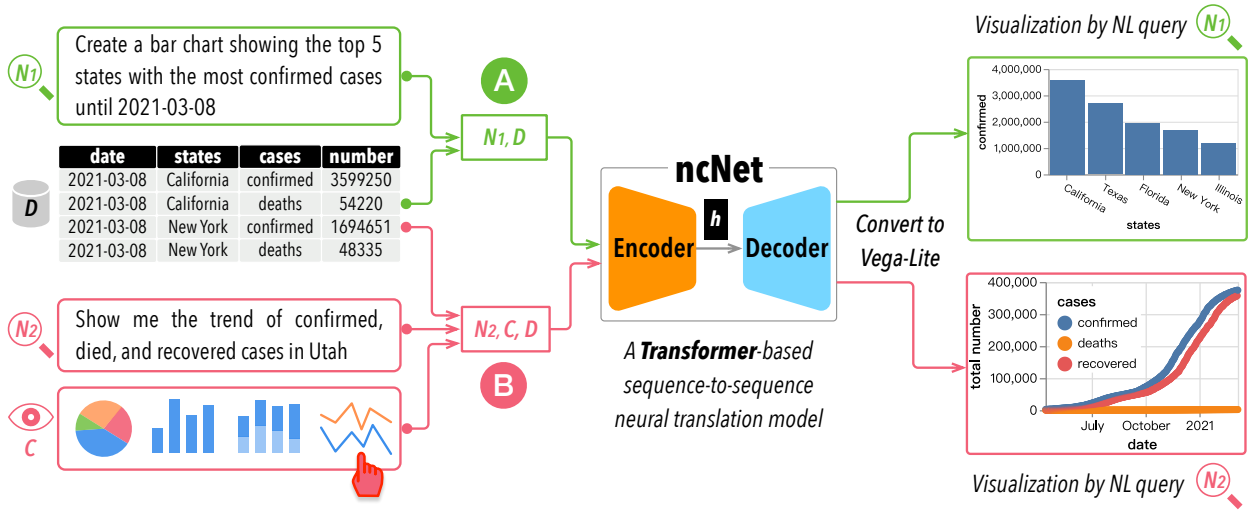


Fig. 1: We present **ncNet**, a Transformer-based sequence-to-sequence model that translates natural language queries to visualizations. It works in two modes. (A) It takes a natural language query N_1 and a dataset D as input, translates them (N_1, D) into a visualization rendered in Vega-Lite. (B) Besides a natural language query N_2 and a dataset D, the user can optionally select a chart template C; **ncNet** will translate the given input (N_2, C, D) into a target visualization.

Abstract— Supporting the translation from natural language (NL) query to visualization (NL2VIS) can simplify the creation of data visualizations because if successful, anyone can generate visualizations by their natural language from the tabular data. The state-of-the-art NL2VIS approaches (*e.g.,* NL4DV and FlowSense) are based on semantic parsers and heuristic algorithms, which are not end-to-end and are not designed for supporting (possibly) complex data transformations. Deep neural network powered neural machine translation models have made great strides in many machine translation tasks, which suggests that they might be viable for NL2VIS as well. In this paper, we present **ncNet**, a Transformer-based sequence-to-sequence model for supporting NL2VIS, with several novel visualization-aware optimizations, including using attention-forcing to optimize the learning process, and visualization-aware rendering to produce better visualization results. To enhance the capability of machine to comprehend natural language queries, **ncNet** is also designed to take an optional chart template (*e.g.,* a pie chart or a scatter plot) as an additional input, where the chart template will be served as a constraint to limit what could be visualized. We conducted both quantitative evaluation and user study, showing that **ncNet** achieves good accuracy in the **nvBench** benchmark and is easy-to-use.

Index Terms—Natural language interface; data visualization; neural machine translation; chart template;

1 Introduction

Natural language interface is a promising interaction paradigm for simplifying the creation of visualizations [32, 43, 52]. If successful, even novices can generate visualizations simply like a Google search. Not surprisingly, both commercial vendors (*e.g.,* Tableau's Ask Data [46], Power BI [2], ThoughtSpot [3], and Amazon's QuickSight [1]) and academic researchers [7,13,20,33,34,40,42,45,49,50,57] have investigated to support the translation from NL queries to visualizations (NL2VIS).

- *Yuyu Luo, Guoliang Li, Chengliang Chai, Xuedi Qin are with the Department of Computer Science, Tsinghua University, China. Email: {luoyy18@mails., liguoliang@, ccl@, qxd17@mails.}tsinghua.edu.cn*
- *Nan Tang is with QCRI, Hamad Bin Khalifa University, Qatar. Email: ntang@hbku.edu.qa*
- *Jiawei Tang is with American School of Doha, Doha, Qatar. Email: 23jtang@asd.edu.qa*
- **Guoliang Li is the corresponding author.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

NL2VIS needs both natural language understanding that uses machines to comprehend natural language queries, and translation algorithms to generate targeted visualization using a visualization language. Natural language understanding is considered an AI-hard problem [56], with many intrinsic difficulties such as ambiguity and underspecification. Many tools from the NLP community, especially based on statistical phrase-based translation [26] and neural machine translation [4, 10], have been used to tackle NL2VIS.

The state-of-the-art NL2VIS methods (for example, NL4DV [40] and FlowSense [57]) are statistical phrase-based translation, which treats natural language understanding and machine translation as two steps. They first employ NLP toolkits (for example, NLTK [5], Stanford CoreNLP [37], and NER [12]) to parse an NL query and produce a variety of linguistic annotations (for example, parts of speech, named entities, etc), based on which they then devise algorithms to generate target visualizations. They are good choices when there are not many training datasets to train deep learning models.

We present **ncNet**, an end-to-end solution using a Transformer-based sequence-to-sequence (seq2seq) model, which translates an NL query to a visualization. It adopts self-attention to generate a rich representation (high dimensional vectors) of the input, **ncNet** enables smart

visualization inference (*e.g.,* guessing the missing column, selecting a chart type, etc).

Besides making smarter inferences, a system can obtain more information (or "hint") from the user, by either obtaining a one-shot hint from the user or iteratively requiring more information (*a.k.a.* conversational systems) [6]. The hint can be of various formats, such as NL queries, tables, chart templates, with one main criterion to be easy-to-use. We propose to use chart templates as additional hints, where a user can specify the output to be a pie chart or a scatter plot with a simple click. In practice, chart templates have been widely used in all commercial products, including Tableau, Excel, Google Sheets, and so on. Due to the flexibility of the seq2seq model, we just treat the selected chart template C as another sequence, together with the NL query N and the dataset D as the input $\mathbf{X}$.

Contributions. In this work, we make several contributions, including:

- proposing **ncNet**, a Transformer-based [53] seq2seq model for supporting NL2VIS;
- presenting a novel visualization-grammar, namely Vega-Zero, with the main purpose to simplify the NL2VIS translation using neural machine translation techniques. Moreover, transforming it to other visualization languages are straightforward;
- enhancing **ncNet** by allowing the user to select a chart template, which will be used to improve the translation accuracy;
- devising two optimization techniques: **attention forcing** for incorporating pre-defined domain knowledge and **visualization-aware translation** for better final visualization generation; and
- demonstrating that **ncNet** can well support NL2VIS with several use cases, as well as conducting a quantitative study.

2 Related Work

2.1 Natural Language Interface for Data Visualization

The idea of using NL as a way to create visualizations was explored around two decades ago [6], where the system interacts with the user through dialogs. During each interaction, the system tries to clarify a small part of the user specification. For example, the system asks: "*At what organizational level?*", the user answers: "*At the department level*", and so on. At that time, the system can only map *simple* user inputs to pre-defined commands.

Afterwards, semantic parsers (*e.g.,* NLTK [5], NER [12], and Stanford CoreNLP [37]), which can automatically add additional layers of semantic information (*e.g.,* parts of speech, named entities, coreference, etc) to NL, have been widely adopted in the research of NL2VIS. Recent studies, such as NL4DV [40] and FlowSense [57], all employ semantic parsers, which are considered as the state of the art.

2.2 Natural Language Processing with Deep Learning

Closer to this work is ADVISor [27] that uses BERT [10] to generate the embeddings of both the NL query and the table headers, which are then used by an "Aggregation" network to select an aggregation type and a "Data" network to decide used attribute and predicates – these SQL fragments will determine an SQL query. Then, a rule-based "Visualization" module will decide which visualization to generate. Compared with ADVISor, **ncNet** supports more complex data transformation types such as relational **join**, **GroupBY**, **OrderBY**, **Or** predicate in SQL WHERE clauses. Another difference is that the neural networks of ADVISor are trained using (NL, SQL) pairs, while **ncNet** is trained using (NL, VIS) pairs and outputs Vega-Zero queries.

In fact, the main obstacle of using deep learning for NL2VIS is not the shortage of deep learning models or techniques. Instead, it is the lack of benchmark datasets that these models can be trained on, because deep learning models are known to be data hungry [14]. Fortunately, a recent work releases the first public benchmark for NL2VIS, namely **nvBench** [35], which can be used to try deep learning for NL2VIS. **nvBench** consists of 25,750 NL queries and the corresponding visualizations, *i.e.,* 25,750 (NL, VIS) pairs, over ∼780 tables from 105 domains (*e.g.,* sports, customers). We will discuss more details of **nvBench** in Section 6.2. Another recent work [48] collected 893 NL queries over three datasets. However, its number is not sufficient to train typical deep learning models.

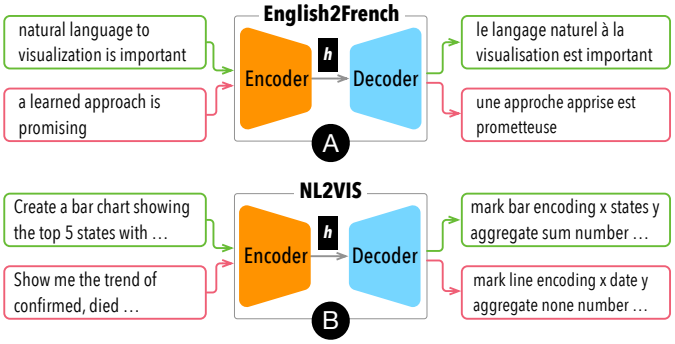


Fig. 2: Sample seq2seq tasks. (A) Translation from English to French. (B) Translation from NL queries to visualization specifications.

An alternative solution is NL2SQL + automatic data visualization, which is a good choice when the entire pipeline is one-shot. However, in practice, it is always iterative. That is, if the target visualization needs to be refined, the user needs to verify/refine both NL2SQL and check the result of automatic data visualization. Note that, checking whether a table is good enough is hard, even for a small table with hundreds/thousands of tuples. In this case, using end-to-end NL2VIS has an advantage that the user only sticks to one task, which is more user-friendly.

3 Design Requirements

There are three main goals when devising solutions for NL2VIS, along the same line of other NL2VIS tools *e.g.,* NL4DV [40].

(1) Easy-to-use. We want to allow novices to create visualizations simply like a Google-search. That is, even users without data visualization background can easily generate visualizations.

(2) End-to-end. Traditional semantic parser based translation systems typically consist of many small sub-components that are tuned separately. In contrast, we want to deliver a complete NL2VIS solution without the need of any additional steps. Besides the well-known benefits of end-to-end solutions such as increased efficiency, cost cutting and ease of learning, one particular benefit for a seq2seq model is that it is easy to maintain and upgrade. For example, upgrading a seq2seq model from using long short-term memory (LSTM) [19] to Transformer [53] only requires to change a few lines of code.

(3) Language-agnostic. The main benefit to be language-agnostic is that we just need to train one seq2seq model for NL2VIS, but can support multiple target visualization languages. The practical need for this is evident, because the users might use various visualization languages constrained by different applications, such as Vega-Lite, D3, ggplot2, and so forth.

4 Background and Problem Formulation

4.1 Sequence-to-Sequence Models

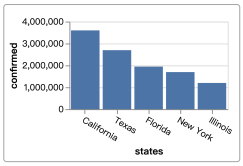
A sequence-to-sequence (seq2seq) model [51] consists of two parts, an encoder and a decoder, where each part can be implemented by different neural networks. The task of an encoder is to understand the input sequence, and generate a smaller representation h (*i.e.,* a high-dimensional vector) to represent the input. The task of a decoder is to generate a sequence of output by taking h as input. The network needs to be trained with a lot of training data, in the form of (Input sequence, Output sequence) pairs.

Due to the flexibility of seq2seq models that allow the input and output to have different formats, they have a wide spectrum of applications including language translation [17], image captioning [38], conversational models and text summarization [39], and NL to SQL [23].

Let's first walk through a typical translation task – language translation from English to French (Figure 2(A)). The task is to train an **English2French** network with a lot of

date	states	cases	number
2021-03-08	California	confirmed	3599250
2021-03-08	California	deaths	54220
2021-03-08	New York	confirmed	1694651
2021-03-08	New York	deaths	48335

(a) A sample table **US_States** of COVID-19.



(b) A bar chart.

Create a bar chart showing the top 5 states with the most confirmed cases until 2021-03-08

(c) A natural language query.

```
mark      bar
data      US_States.csv
encoding  x states y aggregate sum number
transform filter date = "2021-03-08" and cases = "confirmed"
          group x sort y desc topk 5
```

(d) A Vega-Zero specification of the bar chart.

```
{
  "data": {"url": "US_States.csv"},
  "mark": "bar",
  "transform": [{
   "window": [{ "op": "rank", "as": "rank" }],
    "sort": [{
      "field": "number", "order": "descending"}]
    }, {
    "filter": {
      "and": [
        {"field": "datum.rank", "lte": 5},
        {"field": "cases", "equal": "confirmed"},
        {"field": "date", "equal": "2021-03-08"}
      ]
    }}
  ],
  "encoding": {
    "x": {"field": "states", "type": "nominal"},
    "y": {"field": "number", "type": "quantitative",
          "aggregate": "sum"}
   }
}
```

(e) A Vega-Lite specification of the bar chart.

Fig. 3: Sample input, output, and the corresponding Vega-Lite query.

(English sentence, French sentence) pairs, such that it learns to translate from an English sentence (*e.g.,* "natural language to visualization is important") to a French sentence (*e.g.,* "le langage naturel à la visualisation est important").

Similar to the translation from English to French, the translation from NL2VIS is to train a **NL2VIS** network with a lot of (NL query, VIS query) pairs, such that it learns to translate from a natural language query (*e.g.,* "Create a bar chart showing the top 5 ..." in Figure 2(B)) to a visualization specification (*e.g.,* "mark bar encoding x states y aggregate ...").

4.2 Datasets and Natural Language Query

Dataset. Let D be the data source that the user wants to generate visualizations on. To be simple, we consider D as a tabular dataset, which can be obtained from a JSON file, a CSV file, or a relational table from a database.

For example, Figure 3(a) shows a sample data source about the COVID-19 statistical data in the United States.

Natural Language Query. Let N be an NL query that specifies what a user wants to visualize on the dataset D.

For example, Figure 3(c) is an NL query over the dataset in Figure 3(a), which corresponds to a histogram depicted in Figure 3(b).

4.3 Vega-Zero

As we have discussed in Section 4.1, the task of a seq2seq model is to translate an NL sequence to a visualization sequence, for which we need to decide which visualization grammar to use for the output visualization sequence.

Intuitively, we can use Vega-Lite [44]. However, it is hard to train a seq2seq model to generate a hierarchical output (*e.g.,* in JSON format such as Vega-Lite). In contrast, it is much easier to train a seq2seq model to generate a sequence output.

To this end, we present a visualization grammar by simplifying Vega-Lite, with the main purpose to *flatten* a hierarchical Vega-Lite specification to a *sequence*-based specification. That is, Vega-Lite is more **user friendly**, but Vega-Zero is more seq2seq **model friendly**.

Vega-Zero. Vega-Zero keeps most of the keywords of the Vega-Lite about the mapping between visual encoding channels and (transformed) data variables. It flattens a JSON object into a sequence of keywords by removing structure-aware symbols such as brackets, colons, and quotation marks. Formally, a unit specification in Vega-Zero is a four-tuple (similar to Vega-Lite but with each tuple being a sequence) as:

unit = (mark, data, encoding, transform)

Naturally, as a simplification of Vega-Lite: **mark** denotes the chart type, including *bar, line, point (for scatter chart), arc (for pie chart)*; **data** specifies the source data; **encoding** contains x/y-axis, aggregate function, and *color* based on which column; and **transform** defines some data transformation functions: *filter*, *bin*, *group*, *sort*, and *top-k*.

For example, Figures 3(d) and (e) show the Vega-Zero grammar and the Vega-Lite grammar for the target histogram in Figure 3(b), respectively. It shows that Vega-Zero flattens the keywords of the Vega-Lite hierarchical JSON specification into a sequence.

Vega-Zero is Language-Agnostic. Although Vega-Zero is a simplification of Vega-Lite, we still consider it as *language-agnostic*, or an intermediate visualization language, because we remove all language-specific settings such as which color to use (*e.g.,* black or blue), what is the default width of a bar, and so on.

Converting a Vega-Zero query into the format of other visualization languages (*e.g.,* Vega-Lite and ggplot2) is straightforward (see Section 5.5).

4.4 Chart Templates

Note that the reason to allow a user to select a chart template is to alleviate the ambiguity and underspecification intrinsic to an NL query. Besides being user friendly, it can also help novices or data scientists that are not in the visualization community to easily specify the type of desired visualization.

Chart Templates. Figure 4 showcases seven chart templates. Each chart template constraints the chart type, the data types of the x/y-axis, and optional sorting parameters.

For example, if a user selects a "Bar Chart" and then sets the sorting parameter as "by measure attributes in descending order", it means that the user wants to visualize a bar chart and display the bars from high to low, *e.g.,* Figure 3(b).

Essentially, the chart template is used as a **constraint** to reduce the search space of possible outputs. More specifically, a chart template is selected to explicitly constrain the **mark** and *sort* part of a Vega-Zero.

For example, Figure 5(a) shows an "empty" Vega-Zero template without any constraints *w.r.t.* the type of chart, the dataset to use, how to encode, and how to transform. Assume that a bar chart is selected together with a source CSV file "*US_States.csv*" and sorting parameter as "by measure attributes in descending order", then they can be used to fill the **mark** as *bar*, **data** as "*US_States.csv*", and *sort* as *y desc*.

4.5 Problem Statement

Input. Let N be a user provided NL query, C an optional chart template, and D a source dataset. For deep learning, each input needs to be pre-processed before being fed into a deep learning model, typically in the form of vectors [14]. Let $\mathbf{X}_{1:n} = \texttt{embed}(N, C, D)$ be a sequence of n vectors after pre-processing the three inputs N, C and D (see Section 5.2 "Input Embedding" for more details).

Output. Let $\mathbf{Y}_{1:m}$ be the target visualization *w.r.t.* the given input, which is a sequence of m vectors.

We will simply write $\mathbf{X}_{1:n}$ as $\mathbf{X}$ and $\mathbf{Y}_{1:m}$ as $\mathbf{Y}$, when it is clear from the context.

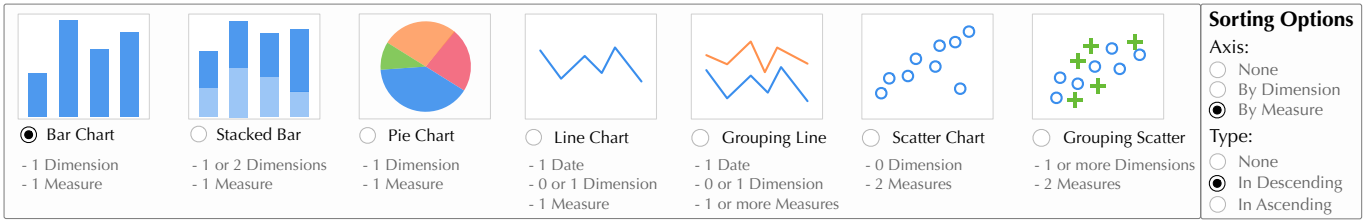


Fig. 4: The chart templates that users can select from.

```
mark      [T]
data      [D]
encoding  x [X] y aggregate [AggFunction][Y] color [Z]
transform filter [F] group [G] bin [B] sort [S] topk [K]
```

(a) An empty chart template in Vega-Zero

```
mark      bar
data      US_States.csv
encoding  x [X] y aggregate [AggFunction][Y] color [Z]
transform filter [F] group [G] bin [B] sort y desc topk [K]
```

(b) A selected chart template in Vega-Zero

Fig. 5: Using a chart template as a constraint.

Problem. The problem of using a `seq2seq` model for `NL2VIS` is to learn a function $f_\theta()$ such that $\mathbf{Y} = f_\theta(\mathbf{X})$, where f_θ is a deep neural network parameterized by θ.

More specifically, a `seq2seq` model consists of two neural networks, an encoder that understands the input sequence and create a hidden representation h (*i.e.,* a high-dimensional vector), and a decoder that generates a target output from h. In a finer granularity, the problem of learning $f_\theta()$ is to learn two connected networks: the encoder network $h = \texttt{encode}(\mathbf{X})$, and the decoder network $\mathbf{Y} = \texttt{decode}(h)$.

5 ncNet

In this paper, we propose to use a Transformer-based `seq2seq` model for `NL2VIS`, namely **`ncNet`**, where "**`n`**" stands for natural language, "**`c`**" for chart template. and "**`Net`**" for neural networks.

5.1 Architecture

The architecture of **`ncNet`** is shown in Figure 6(a). **`ncNet`** adopts a Transformer-based [53] encoder-decoder model that consists of an encoder and a decoder, which are both stacks of self-attention blocks. That is, an encoder (or decoder) is composed of multiple encoder (or decoder) blocks, where an encoder block is depicted in Figure 6(c).

Why Transformer. In retrospect, recurrent neural network (RNN) [15] has been widely used as encoder/decoder blocks. The key innovation of Transformer [53] is that it can process the entire input sequence $\mathbf{X}_{1:n}$ of variable length n without exhibiting a recurrent structure as RNN does, which allows Transformer-based encoder-decoder models to be highly parallelizable. Moreover, this allows the encoder to compute a better contextualized encoding (*e.g.,* when processing a token $\mathbf{x}_i$, it can attend on any input token before or after $\mathbf{x}_i$). In contrast, an RNN can only process tokens sequentially, *i.e.,* when processing a token $\mathbf{x}_i$, it can only see the input tokens before $\mathbf{x}_i$. Due to the advantage of Transfomer to process the input holistically, not sequentially, it can successfully support a wide range of applications including `NL2SQL`, which suggests that it may also be a good fit for `NL2VIS` in the current age. Moreover, explainability on Transformers for neural machine translation has been studied [25], which can help explain how it works to the users. Another advantage is that we can further pre-train Transformers on a large set of `NL` corpus such as Wikipedia such that it can obtain general knowledge such as "MA" in "Cambridge, MA, USA" is a state, and then fine-tune on `NL2VIS` benchmarks.

5.2 The Working Mechanism of ncNet

Next, let's work through an example to understand how **`ncNet`** works.

Consider the three inputs: the `NL` query N, the selected chart C, and the table D as shown in Figure 6(a).

Note that, if the chart template is not specified, we will use the "empty" chart template (see Figure 5(a)) as the input sequence; otherwise, the chart template is selected, we will use a a partially specified Vega-Zero specification as the input sequence (*e.g.,* see Figure 5(b)).

Input Tokenization. We have three types of inputs. Each one will be converted into a sequence, and the three sequences will be concatenated.

1. N: The `NL` query is naturally a sequence of words, where each word will be treated as a token. We add a special token ⟨N⟩ (or ⟨/N⟩) to denote the start (or end) of an `NL` sequence. For example, the `NL` query will be tokenized as:

   $T_N =$ ⟨N⟩ show me the trend of confirmed ... in utah ⟨/N⟩

2. C: Note that there are several tokens such as `[X]`, `[Y]`, `[F]`, `[G]` (see Figure 5) – these are called *masked tokens*. The main task of **`ncNet`** is to fill these masked tokens. Moreover, we add a special ⟨C⟩ (or ⟨/C⟩) token to denote the start (or end) of a chart sequence. For example, the selected line chart is tokenized as:

   $T_C =$ ⟨C⟩ mark line encoding x [x] y aggregate ... ⟨/C⟩

3. D: For simplicity, we consider a single table, which consists of a set of rows with columns. We linearize the D into a sequence of tokens by concatenating the column names and scanning the table content row by row. Similarly, we add a special ⟨D⟩ (or ⟨/D⟩) to denote the start (or end) of a table, the columns (or cell values) tokens will be filled between special token ⟨COL⟩ (or ⟨VAL⟩) and ⟨/COL⟩ (or ⟨/VAL⟩). For example, the table will be tokenized as:

   $T_D =$ ⟨D⟩ table name ⟨COL⟩ ... ⟨/COL ⟩ ⟨VAL⟩ ... ⟨/VAL⟩ ⟨/D⟩

The above three tokenized sequences will be concatenated as one sequence $X = T_N \oplus T_C \oplus T_D$, which is the input of the "Input Embedding" model, shown in Figure 6(b).

Input Embedding. Each token x_i in the input token sequence X will be converted into a vector embedding $\mathbf{x}_i$ as follows:

1. *Token embedding:* Each x_i will be converted to a token embedding, which produces a vector, denoted as $\mathbf{x}_i^{\text{token}}$.
2. *Type embedding:* Each x_i will also be converted to a type embedding, which is responsible for distinguishing the type of token, *e.g.,* an `NL` token, a chart template token, or a data token. It also produces a vector, denoted as $\mathbf{x}_i^{\text{type}}$.
3. *Position embedding:* Finally, we also compute for each x_i a positional embedding, which hints the position of the token x_i in the sequence, denoted as $\mathbf{x}_i^{\text{position}}$.

Finally, the input embedding of x_i, denoted as $\mathbf{x}_i$, is the summation of the above three embeddings as:

$$\mathbf{x}_i = \mathbf{x}_i^{\text{token}} + \mathbf{x}_i^{\text{type}} + \mathbf{x}_i^{\text{position}} = \texttt{embed}(x_i)$$

After each token is processed as discussed above, it outputs $\mathbf{X}$ as shown in Figure 6(b), *i.e.,* $\mathbf{X} = \texttt{embed}(N, C, D)$. This $\mathbf{X}$ will serve as the input to the encoder as shown in Figure 6(a).

Discussion: Handling Large D. When D is small, we can tokenize the entire D as a sequence as discussed above. However, when D is large,

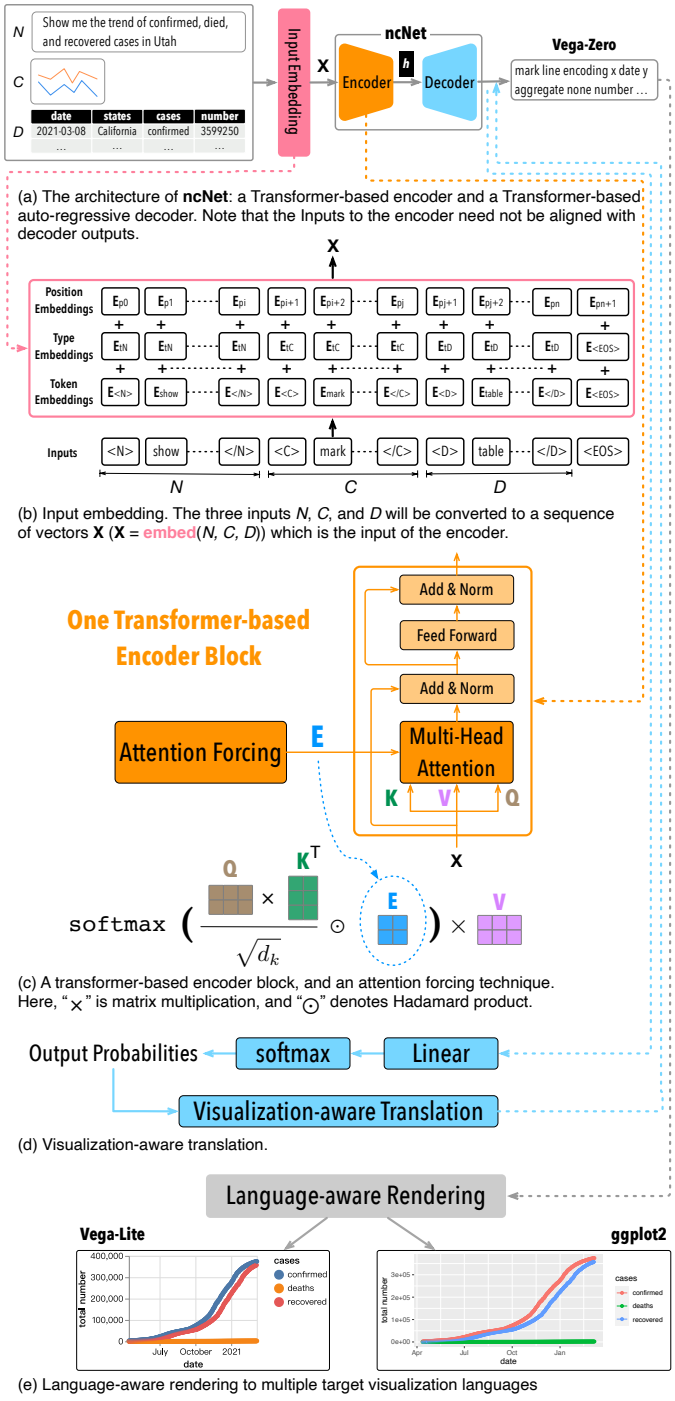


Fig. 6: The architecture of **ncNet**.

due to the limitation of the length of the input vector of a Transformer (*e.g.,* up to 512 tokens), it is not allowed to incorporate all values of D into the sequence. We propose to only add those values that are very possibly mentioned by the user input NL query to the sequence. To this end, we first measure the similarity between a text value in the NL query N and values in D, which is achieved by string-based similarity functions [8], either through handcrafted patterns, or by word embeddings [22]. In this paper, we equip an efficient and effective string similarity search algorithm [8] to extract those values in D that are similar to the text values in the user input NL query N, which are typically small enough to be fed to a Transformer.

After the encoder processes the complete input sequence $\mathbf{X}$, it will generate a contextualized encoded sequence h, as $h = \text{encode}(\mathbf{X})$.

Output Sequence Generation. Next we describe how the decoder can *auto-regressively* generate the output (*i.e.,* the output tokens are sequentially generated) and thus a mapping of an input sequence $\mathbf{X}$ to an output sequence $\mathbf{Y}$.

First, the input encoding h together with a special "start-of-sentence" $\langle \text{SOS} \rangle$ vector, *i.e.,* $\mathbf{y}_0$, is fed to the decoder. The decoder processes the inputs h and $\mathbf{y}_0$ to compute the first target vector $\mathbf{y}_1$. The first target vector $\mathbf{y}_1$ = "mark" is selected.

Next, the decoder now processes both $\mathbf{y}_0 = \langle \text{SOS} \rangle$ and $\mathbf{y}_1$ = "mark" to compute the second target vector $\mathbf{y}_2$ (*e.g.,* "line").

The decoder will then process $\mathbf{y}_0, \mathbf{y}_1$ and $\mathbf{y}_2$, and so on, done in an auto-regressive fashion, until a special $\langle \text{EOS} \rangle$ token is generated as a termination condition.

By doing so, it generates an output sequence as $\mathbf{Y} = \text{decode}(h)$.

5.3 Training

ncNet is trained using an existing NL2VIS benchmark called **nvBench**, which consists of the mapping from thousands of (N, D) pairs to visualizations, such as (N_1, D) and the bar chart in Figure 1. We insert the chart template information C into (N, D) as (N, C, D) with two modes: (1) *empty chart template*, as shown in Figure 5(a); and (2) *selected chart template*, as shown in Figure 5(b). In both cases, they are uniformly treated as partially completed visualization specifications with many masked tokens, such as [X], [Y], [F], and [G], to be filled.

The training is to optimize a reconstruction loss – the cross-entropy between the model output (*i.e.,* the predicted visualization) and the ground truth visualization provided by the benchmark.

5.4 Optimization

5.4.1 Attention Forcing

A major innovation of Transformer [53] is **attention**, which allows the model to focus on the relevant parts of the input sequence as needed, and thus highly improves the quality of machine translation systems. Next, let's first review the concept of attention, followed an optimization technique, called **attention forcing**, to further improve the quality.

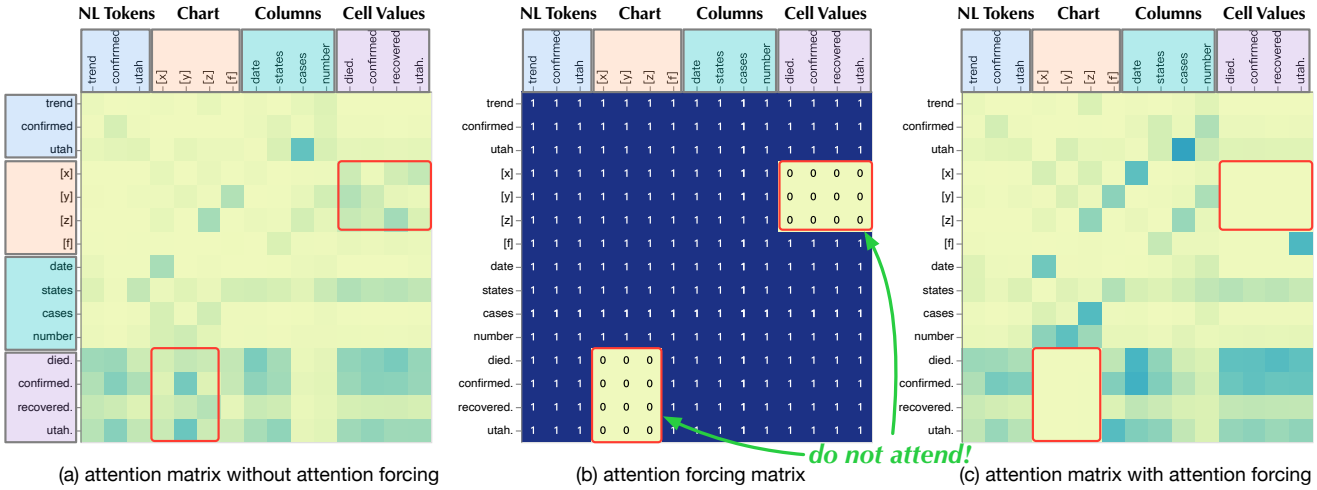
Let's Pay "Attention". Figure 6(c) shows a Transformer-based encoder block, where the "Multiple-Head Attention" module gives the encoder greater power to estimate the relevance of one token to other tokens. For simplicity, in the following, we only illustrate "One-Head Attention" (*i.e.,* one attention unit), and "Multiple-Head Attention" just repeats the computation of one attention unit multiple times in parallel. A sample attention matrix is shown in Figure 7(a), which shows the encoder attention weights of pairs of tokens in the range $[0, 1]$ that correspond to colors from light chartreuse to dark blue. A light color (or a dark color) indicates that the corresponding two tokens are less (or more) correlated.

Next we briefly discuss how an attention matrix is computed (see Figure 6(c)). Recall the input $\mathbf{X}$ is a sequence of vectors, which forms a matrix by packing these vectors. The Transformer will learn three weight matrices: the query weight matrix $W^{\mathbf{Q}}$, the key weight matrix $W^{\mathbf{K}}$ and the value weight matrix $W^{\mathbf{V}}$. It then computes three matrices as $\mathbf{Q} = \mathbf{X} \times W^{\mathbf{Q}}$, *i.e.,* a matrix that contains the query (vector representation of one word in the sequence); $\mathbf{K} = \mathbf{X} \times W^{\mathbf{K}}$, *i.e.,* a matrix with all the keys (vector representations of all the words in the sequence); and $\mathbf{V} = \mathbf{X} \times W^{\mathbf{V}}$ *i.e.,* a matrix with all the values, which are again the vector representations of all the words in the sequence, where "×" means matrix multiplication. The three matrices, $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$, will be fed to the attention unit. The output attention matrix is computed as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}}\right) \times \mathbf{V},$$

where $\mathbf{K}^T$ is the transpose of matrix $\mathbf{K}$, $\sqrt{d_k}$ is to stabilize gradients during training, and softmax normalizes the weights to sum to 1. Please refer to the Transformer paper [53] for more details.

Attention Forcing. It can be observed from Figure 7(a) that there is no need to have an attention value between [X] and cell values *recovered, confirmed, died, and utah*, because these cell values should not be encoded as the x-axis of a visualization. If we have such prior



(a) attention matrix without attention forcing (b) attention forcing matrix (c) attention matrix with attention forcing

Fig. 7: Examples of attention forcing. Plotting attention scores for all tokens are cluttered, we sample some tokens for visualization.

knowledge, we can explicitly tell the attention unit about *what should not be attended*, which is known as *attention forcing* [11].

More specifically, we explicitly encode the prior knowledge as a **forcing matrix E** with boolean values 0 and 1, where 0 means *do not attend*. In the case of `NL2VIS`, the values (*i.e.,* `[X]/[Y]/[Z]`) of the *encoding* part, grouping `[G]`, binning `[B]`, and sorting `[S]` should be selected from the *columns* of the relational table D, while the values (*i.e.,* `[F]`) of the *filter* part can be filled by both the *columns* and *cell values* of the D.

A sample forcing matrix is shown in Figure 7(b), which shows that it does not attend the tokens between `[X]/[Y]/[Z]` and cell values.

Next, we discuss how to force the attention to be aware of the forcing matrix **E**. Essentially, the forcing matrix **E** acts as an attention mask so that some attention scores of the irrelevant tokens are set to zero. This is achieved by an element-wise multiplication as below:

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V},\mathbf{E}) = \text{softmax}\left(\frac{\mathbf{Q}\times\mathbf{K}^T}{\sqrt{d_k}}\odot\mathbf{E}\right)\times\mathbf{V},$$

where "$\odot$" denotes Hadamard product.

The process of attention forcing is also depicted in Figure 6(c) to help illustrate the difference with/without the forcing matrix **E**.

The computed attention matrix, after attention forcing, is shown in Figure 7(c). Comparing with Figure 7(a), we can see that the attention scores in red rectangles of Figure 7(c) are set to zero. In contrast, the attention scores of (`[X]`, date) , (`[Y]`, number), (`[Z]`, cases), (`[F]`, states), (`[F]`, utah) become higher, which indicate that the model better capture token correlation after attention forcing.

5.4.2 Visualization-aware Translation

In the decoding phase, the model may make false prediction due to various reasons. We detail some reasons as follows. The first reason is the inherent ambiguity of the `NL` query. The second reason is that the `NL2VIS` model may work poorly in some situations. For example, when it predicts the *encoding* parts, the tokens should be selected from the column names of the table D instead of the cell values of the D.

To alleviate the above issues, we propose a simple yet effective method, namely **visualization-aware translation**, that incorporates visualization knowledge (*i.e.,* rules of thumb for visualization) and chart templates to validate the output tokens of the **`ncNet`** and correct those incorrect tokens (Figure 6(d)). Our visualization-aware translation method is a variant of the *Beam Search* [51], a well-known algorithm for neural machine translation task. In a nutshell, the beam search algorithm [47] maintains a set of k candidate partial outputs and their cumulative probabilities at each step, and finally select k candidates with highest cumulative probabilities.

The basic idea of **visualization-aware translation** is to maintain the top-k possible tokens predicted by the **`ncNet`** at some *specific steps*, and then we use visualization rules and the constraints of a selected chart template to select the "*most possible*" token from the top-k candidates. More concretely, the *specific steps* are those steps for predicting the empty parts of the Vega-Zero grammar (*e.g.,* the `[T]`, `[X]` in Figure 5(a)). For steps of predicting the following types of tokens, by default we maintain top-5 tokens given by the model, and choose the "best" with the following *heuristic rules*:

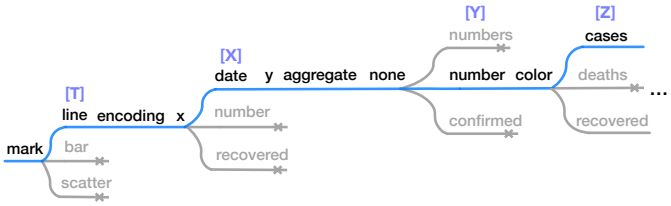


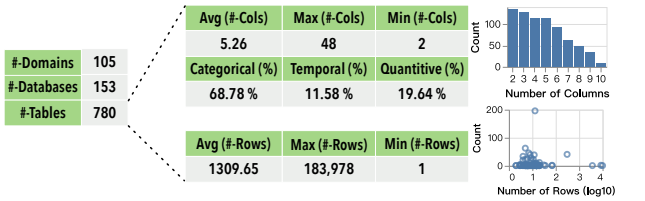
Fig. 8: An example of visualization-aware translation. It shows how to select the "best" token from top-3 candidates. The blue path is selected by our algorithm, while the gray paths are ignored by the algorithm. Tokens in specific steps are listed by the probability in descending order. It selects the "most possible" token in specific steps (*e.g.,* line for `[T]`).

1. For the `[T]/[S]` tokens: if a chart template is specified, we use the parameters from the chart template as the target outputs; otherwise, we take the top-1 candidate as the output token.
2. For the `[X]/[Y]/[Z]` tokens: we choose the candidate token t with the highest probability under the condition that the t is a column of the dataset D and the column type should suitable for the chart type, *e.g.,* two quantitative columns for scatter chart.
3. For the `[G]` (or `[B]`) tokens: for the group (or bin) key, we choose the candidate token t with the highest probability and it must satisfy: (i) the t is temporal/categorical (or temporal/quantitative) types, and (ii) the t should not be the tokens from the `[Z]` part.
4. For the `[K]` token: we keep the candidate token t with the highest probability under the condition that the t is a valid number.

For example, Figure 8 shows an example to demonstrate the working mechanism of the **visualization-aware translation** algorithm. The algorithm follows *heuristic rules* mentioned before to process output tokens in those specific steps. For example, for the `[T]` token, since the token "line" has the highest probability and is a right chart type specified by the chart template, the algorithm keeps it as the target output. For the `[Y]` token, it selects the rank-2 token "number" because the rank-1 token "numbers" is not a column of the dataset D.

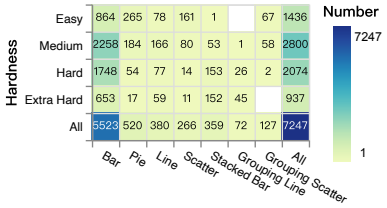
Remark: Visualization Recommendation. Although the above process will output only one visualization, it is readily to be extended to provide top-k visualizations, *e.g.,* by ranking different branches in Figure 8, which can be used for visualization recommendation.

We have empirically verified that using this optimization can increase ~3% accuracy, which will be used by **`ncNet`** by default.



(a) NVBench data source statistics

VIS Types	#-VIS	#-(NL, VIS)
Bar Chart	5523	19407
Pie Chart	520	1750
Line Chart	380	1562
Scatter Chart	226	1041
Stacked Bar	359	1172
Grouping Line	72	271
Grouping Scatter	127	547
All Types	7247	25750



(b) NVBench natural language query and visualization

Fig. 9: The statistics of the **nvBench** benchmark.

5.5 Language-aware Rendering

As shown in Figure 6(e), the Vega-Zero specification can be converted to popular visualization languages (*e.g.,* Vega-Lite, ggplot2) for rendering the visualization result. The translation from a Vega-Zero specification to a targeted visualization language is hard-coded. Currently, we write ∼240 and ∼333 lines Python3 code to support the translation from Vega-Zero to Vega-Lite and ggplot2, respectively. The code will be open sourced.

6 Implementation

6.1 Implementation Details of ncNet

We implement the **ncNet** with PyTorch [41], with three Transformer-based encoder blocks and three Transformer-based decoder blocks. For the *multi-head attention layer*, we set the number of heads to 8. The embedding dimension for word embedding, token types embedding and position embedding are set to 256. We limit the input length to 512 tokens – inputs that exceed this limit are truncated.

We use the Adam optimizer [24] with a static learning rate (*i.e.,* 0.0005) instead of the one with warm-up and cool-down steps. We use a learned positional encoding. We set the dropout rate as 0.1, for both encoder and decoder. The batch size is set to 64. The size of the trained model is about 40 MB.

6.2 NL2VIS Benchmark

For training and testing **ncNet**, we use a public benchmark for the NL2VIS task, namely **nvBench** [30, 35].

NVBench Statistics. Figure 9 overviews the statistics of **nvBench** from the data sources and (NL, VIS) perspectives. As shown in Figure 9(a), **nvBench** has 153 databases along with 780 tables in total and covers 105 domains (*e.g.,* sports, customers). The average number of columns/rows of the 780 tables is 5.26/1,309.65, and the maximum/minimum number of columns (rows) is 48/2 (183,978/1). Among the columns, 68.78% of columns are categorical columns, 11.58% of columns are temporal columns, and 19.64% of columns are quantitative columns. Figure 9(a) also depicts the distributions of columns and rows, which tells us that most of the tables have 2 to 9 columns.

Given 153 databases, as shown in Figure 9(b), **nvBench** contains 7,274 visualizations on seven types of charts. For each visualization, **nvBench** provides one to several NL queries because different users might provide different NL queries for the same visualization. In total, **nvBench** consists of 25,750 (NL, VIS) pairs. **nvBench** further defines the four-level complexities, *i.e., easy, medium, hard, and extra hard*, of the visualizations based on the hardness of the visualization query (*i.e.,* similar to the Vega-Zero). For example, a Vega-Zero with *filter*, *bin* and *aggregations* may be categorized as a *Hard* visualization. The heatmap in Figure 9(b) shows the distribution of visualizations in different chart types and hardness of visualization queries.

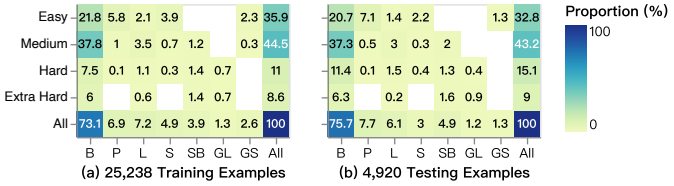


Fig. 10: The distribution of training and test set. B (Bar Chart), P (Pie Chart), L (Line Chart), S (Scatter Chart) SB (Stacked Bar Chart), GL (Grouping Line Chart), GS (Grouping Scatter Chart).

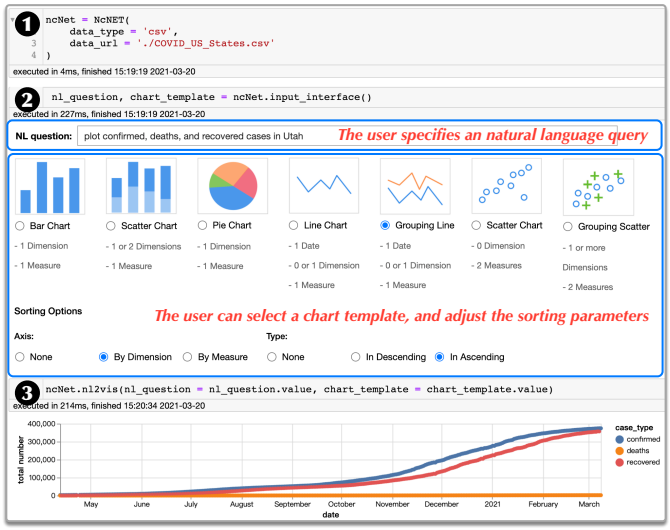


Fig. 11: Using **ncNet** in Jupyter Notebook.

6.3 Training ncNet

nvBench splits the 25,750 (NL, VIS) pairs into a training set with 20,598 pairs, a development set consisting of 1,162 pairs, and a test set with 3,990 pairs. Based on the original training, development, and test sets provided by **nvBench**, we further process them to satisfy our requirement, as discussed below.

First, since **nvBench** contains some visualizations generated from multiple tables with the **join** operations, we remove such cases so as to focus on **non-join** cases in this evaluation. In average, the accuracy **join** cases is lower than **non-join** cases for ∼ 3.6%.

Second, we have to inject the chart templates into the benchmark to train **ncNet** for generating visualization result with NL query and chart template as inputs, because **nvBench** only provides NL query and the corresponding visualization. This step is straightforward: given a (NL, VIS) pair, we assign this (NL, VIS) pair a chart template (*e.g.,* Figure 5(b)) based on its chart type and sorting parameter. Hence, each (NL, VIS) pair will result in two (NL, VIS) pairs, one with an empty chart template, and the other with the desired chart template (*e.g.,* a bar chart with sorting y-axis in descending order).

Finally, we have training/test sets with 25,238/4,920 (NL, VIS) pairs. More specifically, we show the distribution of training and testing (NL, VIS) pairs in Figure 10. We can see that they have similar distributions on chart types and the hardness of visualization queries.

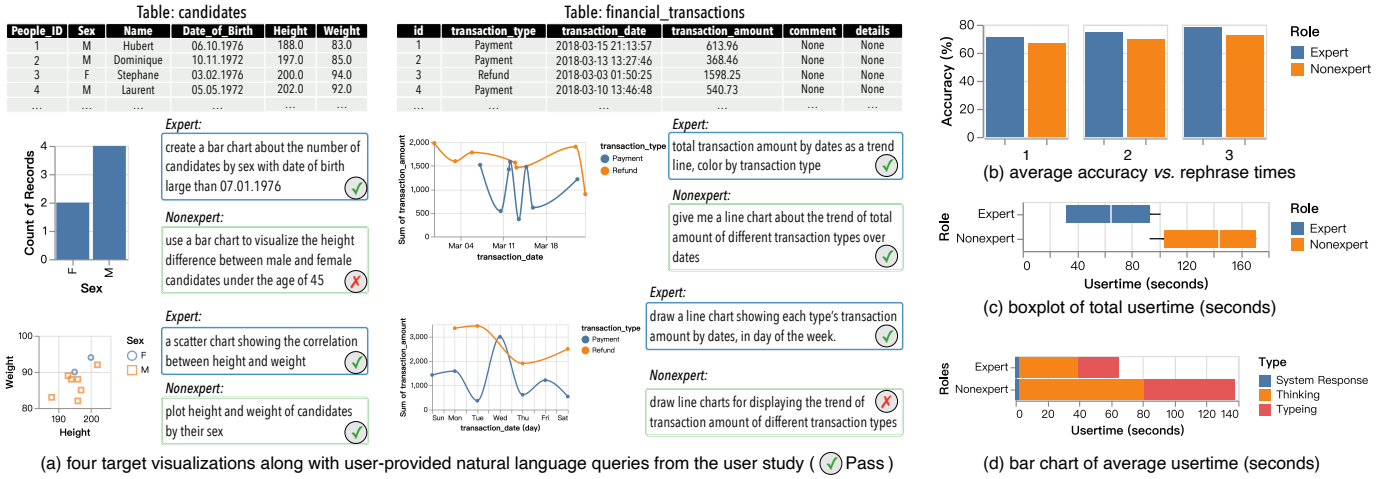
7 Evaluation

7.1 Usage Scenarios

We first present two common usage scenarios to demonstrate how developers (or lay users) can use **ncNet**.

ncNet in Jupyter Lab. Data science practitioners often perform interactive data visualization in Jupyter Lab (or Jupyter Notebook). To make **ncNet** easy-to-use to for this type of users, we have developed a Python package to be used in the Jupyter Lab ecosystem.

Figure 11 is a screenshot that demonstrates how to create a desired *grouping line chart* with an NL query and a chart template for a COVID-19 dataset. First, the user imports the **ncNet** package and



(a) four target visualizations along with user-provided natural language queries from the user study (✓ Pass)

Fig. 12: User study.

creates an instance of **ncNet** by setting the dataset D variables (Figure 11-❶). Next, the user can initialize an input interface for the NL query N and the chart template C (Figure 11-❷). Alternatively, the user can also directly pass the N and C variables to the function `input_interface(nl_query, chart_template)`. Finally, **ncNet** will render the best inferred visualization relevant to N and C in Jupyter Lab using Vega-Lite library (Figure 11-❸).

ncNet as a Toolkit. In addition to using **ncNet** in an interactive Python environment, we also regard **ncNet** as a Python toolkit for the NL2VIS task. It means developers can build their visualization systems or dashboards [28,29,31,36] via using **ncNet** package through pre-defined APIs.

7.2 User Study

We conducted a user study on **ncNet** with two goals: (1) whether **ncNet** works well in real-world datasets with users from multiple domains? and (2) collect qualitative feedback on **ncNet**'s design criterion.

Tasks. To achieve the above goals, we design the following tasks.
Task 1: Given a table and a target visualization, we asked the participants to express an NL query for creating such a visualization.
Task 2: We asked participants to provide their comments about **ncNet**.

Participants. We invited 4 experts (2 VISers, 1 NLPer, and 1 DBer) and 2 non-experts (1 financial staff and 1 operational staff) to participate in our user study. The four experts have expertise in data analysis/visualization and have more than six years of experience in Python, SQL, and Tableau. The latter two users are familiar with Excel and creating simple visualizations (*i.e.,* plotting two columns as a bar chart in Excel).

Procedures. We asked participants to vote for 5 databases with domains they are familiar with out of 153 databases in **nvBench**. The 5 databases selected by majority voting were used in the user study. We then taught participants to use **ncNet** in Jupyter Lab. The experts took about 5 minutes to understand how to use **ncNet**, while the two non-technical participants took about 20 minutes to understand the usage of **ncNet**. Non-technical participants had longer learning time mainly because they need to be familiar with the Jupyter Lab environment, including inputting/rephrasing input (NL queries) and executing cell. Third, we randomly sampled 10 visualizations for each database as the targets. Totally, we have 50 target visualizations for the user study. For task 1, we provided descriptions (*i.e., what* is expected) for each target visualization to guide the participants to formulate their NL queries (*i.e.,* how). We recorded the interaction logs, including NL queries, input records, and task situations, when the participants interacted with **ncNet**. For task 2, we collected their comments about **ncNet**.

Results. Figure 12(a) samples 4 target visualizations with user-provided NL queries. We also denote (using ✔) which NL queries can successfully generate the target visualization using **ncNet**. Figure 12(b) reports the average accuracy; we can see that the experts achieve better results than non-experts. By rephrasing NL query several times, the accuracy rate is also improved. We also analyze the interaction logs from Task 1. Figure 12(c) shows the time for non-experts to perform a NL2VIS task is about twice than experts. We report more details in Figure 12(d). It depicts the composition of a round of user time, in which thinking time and typing time account for almost half, and the system response time is around 1 second.

The participants comment that that **ncNet** is easy-to-use and can generate the target visualization in easily. Specifically, the two non-technical users think that the NL2VIS tool is easier than Excel because it only needs to express the intent of visualization using natural language queries instead of conducting a series of operations in the Excel.

7.3 Quantitative Evaluation

We now present quantitative assessments of **ncNet** using the **nvBench** testing dataset mentioned in Section 6.3. We use *accuracy* as the evaluation metrics. To be conservative, the *accuracy* measures whether the output Vega-Zero sequence exactly matches the ground truth Vega-Zero sequence. We define the *accuracy* $= M/N$, where M is the number of the output Vega-Zero sequences that are equivalent to the ground truth Vega-Zero sequences, and N is the total number of testing cases.

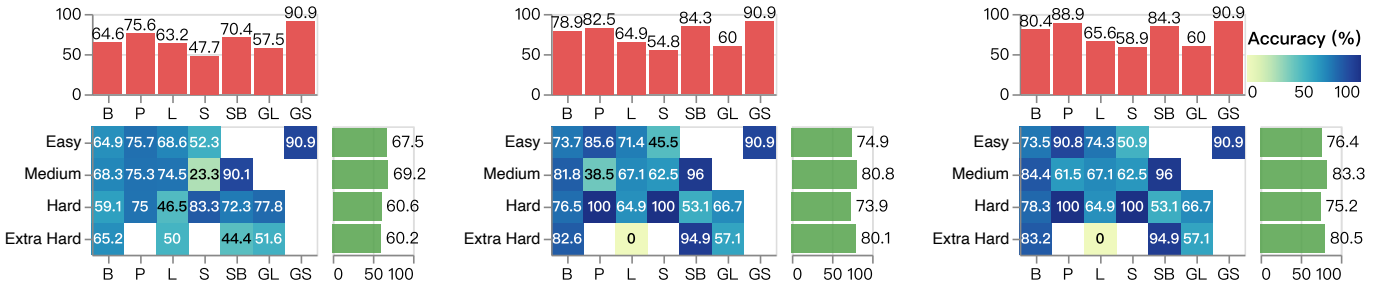
We evaluate the following three cases on the same testing dataset:

(1) *Neural Network-based Baseline*: we adapt the state-of-the-art NL2SQL model [18] for NL2VIS task (Figure 13(a)).
(2) ***ncNet*** *without chart template* (Figure 13(b)).
(3) ***ncNet*** *with a selected chart template* (Figure 13(c)).

Note that, we only consider the first visualization result (*i.e.,* rank-1) outputted by baseline model and **ncNet**. Figure 13 summarizes the evaluation results. More concretely, the heatmap details the accuracy under different visualization types and difficult levels of the NL query. The bar charts colored in red report the average accuracy by chart types, while the green bar charts show the average accuracy across different hardness of NL queries. Next, we will elaborate the evaluation results.

ncNet: Overall Performance. Figures 13(b) and (c) summarize the performance of **ncNet** without and with chart template, respectively. The **ncNet** performs well on different hardness levels of visualization queries (shown in the green bar chart) and works effectively on different types of visualizations (the red bar chart). More specifically, as shown in the heatmaps of Figures 13(b) and (c), we can see that **ncNet** achieves 100% accuracy on some cases (*e.g.,* (S, Hard) in Figure 13(b) and (c)). Overall, **ncNet** shows its effectiveness by achieving the accuracy of 77.8% and 79.6% on average for the cases without and with a chart template, respectively.

ncNet: With/Without Chart Template. Next we give a closer look at the performance difference between two cases of **ncNet** with and without chart template in Figure 13(b) and (c), respectively. In general, it shows that the accuracy of nearly all cases is higher when an



(a) Neural Network-based Baseline (b) **ncNet** w/o Chart Template (c) **ncNet** with Chart Template

Fig. 13: Quantitative Evaluation. B (Bar Chart), P (Pie Chart), L (Line Chart), S (Scatter Chart), SB (Stacked Bar Chart), GL (Grouping Line Chart), GS (Grouping Scatter Chart).

additional chart template is specified, which can be observed from the red and green bar charts in Figure 13(b) and (c) that the accuracy in Figure 13(c) is higher than Figure 13(b). This is obvious especially for *bar*, *pie* and *scatter* charts.

ncNet vs. Neural Network-based Baseline. The evaluation of the neural network-based baseline is shown in Figure 13(a). The baseline solution achieves an average 65.0% accuracy, while our approaches outperform it by 12.8% and 14.6%, respectively. More concretely, we can compare the performances across different solutions in the heatmaps (Figure 13). It shows that **ncNet** is better than the neural network-based baseline in almost all cases. The result is expected, because **ncNet** uses several optimization techniques specifically designed for the task of NL2VIS.

Error Analysis. We conduct an error analysis on those failed cases. There are four main causes of evaluation error of **ncNet**.

(1) Fail on predicting columns or transform parts. This is the main cause of failure. For mispredicted columns, it is mainly because the NL query mentions desired columns in an ambiguous or implicit way. In some cases, it fails due to the "unperfect" table header (*e.g.,* "StuId" *vs.* "Student ID"). Incorporating some pre-trained language models (*e.g.,* BERT [10]) or processing the table headers might help to tackle this issue. The main reason of mispredicting `transform` parts (*e.g.,* filter) is mainly regard to problem of natural language understand. That is how to successfully convert the intend into right logic operations. For example, as shown in Figure 12(a), the NL query mentions "... candidates under the age of 45 ...", it should construct the `[F]` as "... `filter Date_of_Birth > 07.01.1976` ...". The above issues could be alleviated by developing more advanced natural language understanding models and designing smart strategies for jointly understanding the user intends incorporating database information and visual analysis knowledge.

(2) Fail on predicting chart type and sorting parameters. This case usually appears when **ncNet** does not take a chart template as an additional input.

(3) True result in the rank-k output. Since our quantitative evaluation only considers the rank-1 Vega-Zero returned by **ncNet**. However, we observe that the ground truth Vega-Zero can be generated at the rank-*k* (*e.g.,* rank-4) result by **ncNet**, in some cases. Therefore, this observation points out that the performance of NL2VIS can be further improved by incorporating visualization recommendation techniques, *i.e.,* suggesting top-*k* visualization results relevant to the NL query.

(4) Vega-Zero unmatched but visualization result match. Some of predicted Vega-Zero sequences are unmatched with the ground truth Vega-Zero sequences, but their visualization results are in fact equivalent. For example, "`mark bar encoding x cars y aggregate count cars ...`" equals "`mark bar encoding x cars y aggregate count cylinders ...`" because they have the same aggregate results used for showing the number of cars in a bar chart.

8 Limitations and Future Work

8.1 Limitations

(L1) *Limited benchmarks.* In retrospect, benchmarks have played a key role in spawning the boom in different research communities, such as ImageNet [9] for image processing, GLUE [55] and SuperGLUE [54] for the NLP community, and TPC benchmarks [16] for the database community. However, large public benchmarks, such as VizNet [21] and **nvBench** [35], that can be used for deep learning on data visualization tasks are quite limited.

(L2) *Supporting only one-shot NL queries.* Different from applications that only need one-shot queries, data analytics often needs to issue a sequence of (or iterative) queries. End-to-end approaches (*e.g.,* NL2VIS) have the advantage that the user only needs to check the visualization and refine the NL query. In contrast, non end-to-end approaches might require the user to be involved in different tasks, which is less user friendly. Even though, however, our current NL2VIS model only translates one (possibly revised) NL to a visualization, instead of translating a sequence of (or conversational) NL pieces into a visualization.

8.2 Future Work

(F1) *The quest for more benchmarks.* In response to limitation **L1**, a promising future direction for pushing deep learning for data visualization, is to contribute to new benchmarks that cover more diversified tasks, such as conversational NL2VIS benchmarks, annotations for chart to description mapping, annotation for similar charts, and so on.

(F2) *Supporting conversational NL queries.* To lift limitation **L2**, an interesting direction is to extend **ncNet** to support conversational NL queries. The good news is that there are text-to-SQL benchmarks such as CoSQL [58]. Naturally, it is interesting to explore how to leverage these benchmarks to extend and train **ncNet** such that it can support conversational NL2VIS cases, which have lots of practical applications.

(F3) *Chart2vec.* Similar to word2vec that learned a universal embedding of words based on the word associations from a large corpus of text, an interesting future work is to learn a universal embedding of charts, so as to enable other downstream applications (*e.g.,* recommendations, story telling, guideline generation, and so forth.)

9 Conclusion

We present **ncNet**, a research attempt using deep neural networks to support NL2VIS. This approach is built upon the latest NLP models, namely Transformer-based `seq2seq` models. We further propose to use chart templates to help enhance the translation accuracy. We also demonstrate the effectiveness of **ncNet** on a NL2VIS benchmark over 105 domains. We hope that our proposal **ncNet**, along with recent advances in NLP, can shed some light on NL2VIS and justify the potential of deep neural networks for NL2VIS. However, the journey just starts and a lot need to be done such as handling non-friendly column header names (or even foo_bar) and work for big data.

Acknowledgments

This project is supported by NSF of China (61925205, 61632016), Beijing National Research Center for Information Science and Technology (BNRist), Huawei, TAL Education, China National Postdoctoral Program for Innovative Talents (BX2021155), China Postdoctoral Science Foundation (2021M691784), and Zhejiang Lab's International Talent Fund for Young Professionals.

REFERENCES

[1] Amazon's QuickSight, https://aws.amazon.com/cn/blogs/aws/amazon-quicksight-q-to-answer-ad-hoc-business-questions/.

[2] Microsoft Power BI Q&A. https://docs.microsoft.com/en-us/power-bi/create-reports/power-bi-tutorial-q-and-a.

[3] SpotIQ AI-Driven Insignts (2nd Edition). https://www.thoughtspot.com/resources#white_paper.

[4] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[5] S. Bird. NLTK: the natural language toolkit. In N. Calzolari, C. Cardie, and P. Isabelle, eds., *ACL*, 2006.

[6] K. C. Cox, R. E. Grinter, S. Hibino, L. J. Jagadeesan, and D. Mantilla. A multi-modal natural language interface to an information visualization environment. *Int. J. Speech Technol.*, 4(3-4):297–314, 2001.

[7] W. Cui, X. Zhang, Y. Wang, and et al. Text-to-viz: Automatic generation of infographics from proportion-related natural language statements. *IEEE Trans. Vis. Comput. Graph.*, 26(1):906–916, 2020.

[8] D. Deng, G. Li, and J. Feng. A pivotal prefix based filtering algorithm for string similarity search. SIGMOD '14, p. 673–684, 2014.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[10] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pp. 4171–4186.

[11] Q. Dou, J. Efiong, and M. J. F. Gales. Attention forcing for speech synthesis. In *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, pp. 4014–4018. ISCA, 2020.

[12] J. R. Finkel, T. Grenager, and C. D. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *ACL*, pp. 363–370, 2005.

[13] T. Gao, M. Dontcheva, and et al. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *UIST*, 2015.

[14] I. J. Goodfellow, Y. Bengio, and A. C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

[15] A. Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[16] J. Gray. Introduction. In *The Benchmark Handbook for Database and Transaction Systems (1st Edition)*, pp. 1–17. Morgan Kaufmann, 1991.

[17] D. Guo, G. Tür, W. Yih, and G. Zweig. Joint semantic utterance classification and slot filling with recursive neural networks. In *SLT*, 2014.

[18] J. Guo, Z. Zhan, Y. Gao, and et al. Towards complex text-to-sql in cross-domain database with intermediate representation. In *ACL*, 2019.

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[20] E. Hoque, V. Setlur, M. Tory, and I. Dykeman. Applying pragmatics principles for interaction with visual analytics. *IEEE TVCG*, 2018.

[21] K. Z. Hu, S. N. S. Gaikwad, and et al. Viznet: Towards A large-scale visualization learning and benchmarking repository. In *CHI*, 2019.

[22] T. Kenter and M. de Rijke. Short text similarity with word embeddings. In *CIKM*, pp. 1411–1420, 2015.

[23] H. Kim, B. So, W. Han, and H. Lee. Natural language to SQL: where are we today? *Proc. VLDB Endow.*, 13(10):1737–1750, 2020.

[24] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, eds., *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[25] G. Kobayashi, T. Kuribayashi, S. Yokoi, and K. Inui. Attention is not only a weight: Analyzing transformers with vector norms. In *EMNLP*, pp. 7057–7075, 2020.

[26] P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the ACL*, pp. 127–133, 2003.

[27] C. Liu, Y. Han, R. Jiang, and X. Yuan. Advisor: Automatic visualization answer for natural-language question on tabular data. In *14th IEEE Pacific Visualization Symposium, PacificVis 2021, Tianjin, China, April 19-21, 2021*, pp. 11–20. IEEE, 2021.

[28] Y. Luo, C. Chai, X. Qin, N. Tang, and G. Li. Interactive cleaning for progressive visualization through composite questions. In *36th IEEE International Conference on Data Engineering, ICDE*. IEEE, 2020.

[29] Y. Luo, C. Chai, X. Qin, N. Tang, and G. Li. Visclean: Interactive cleaning for progressive visualization. *Proc. VLDB Endow.*, 13(12), 2020.

[30] Y. Luo and et al. Empowering natural language to visualization neural translation using synthesized benchmarks. In *VIS*, 2021.

[31] Y. Luo, W. Li, T. Zhao, X. Yu, L. Zhang, G. Li, and N. Tang. Deeptrack: Monitoring and exploring spatio-temporal data - A case of tracking COVID-19 -. *Proc. VLDB Endow.*, 13(12):2841–2844, 2020.

[32] Y. Luo, X. Qin, C. Chai, N. Tang, G. Li, and W. Li. Steerable self-driving data visualization. *IEEE Transactions on Knowledge and Data Engineering*, 2020. doi: 10.1109/TKDE.2020.2981464

[33] Y. Luo, X. Qin, N. Tang, and G. Li. Deepeye: Towards automatic data visualization. In *ICDE*, pp. 101–112, 2018.

[34] Y. Luo, X. Qin, N. Tang, G. Li, and X. Wang. Deepeye: Creating good data visualizations by keyword search. In *SIGMOD*, 2018.

[35] Y. Luo, N. Tang, G. Li, C. Chai, W. Li, and X. Qin. Synthesizing natural language to visualization (NL2VIS) benchmarks from NL2SQL benchmarks. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. ACM, 2021.

[36] Y. Luo, N. Tang, G. Li, and et al. Deepeye: A data science system for monitoring and exploring COVID-19 data. *IEEE Data Eng. Bull.*, 2020.

[37] C. D. Manning and et al. The stanford corenlp natural language processing toolkit. In *ACL*, pp. 55–60, 2014.

[38] J. Michael, R. Labahn, T. Grüning, and J. Zöllner. Evaluating sequence-to-sequence models for handwritten text recognition. In *ICDAR*, 2019.

[39] R. Nallapati, B. Xiang, and B. Zhou. Sequence-to-sequence rnns for text summarization. *CoRR*, abs/1602.06023, 2016.

[40] A. Narechania, A. Srinivasan, and J. T. Stasko. NL4DV: A toolkit for generating analytic specifications for data visualization from natural language queries. In *VIS*, 2020.

[41] A. Paszke, S. Gross, F. Massa, A. Lerer, and e. a. Bradbury. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. 2019.

[42] X. Qin, Y. Luo, N. Tang, and G. Li. Deepeye: Visualizing your data by keyword search. In *EDBT*, pp. 441–444, 2018.

[43] X. Qin, Y. Luo, N. Tang, and G. Li. Making data visualization more efficient and effective: a survey. *VLDB J.*, 29(1):93–117, 2020.

[44] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE TVCG*, 23(1):341–350, 2017.

[45] V. Setlur, S. E. Battersby, and et al. Eviza: A natural language interface for visual analysis. In *UIST*, 2016.

[46] V. Setlur, M. Tory, and A. Djalali. Inferencing underspecified natural language utterances in visual analysis. In W. Fu, S. Pan, O. Brdiczka, P. Chau, and G. Calvary, eds., *IUI*, pp. 40–51, 2019.

[47] M. Spero. Improved beam search diversity for neural machine translation with k-dpp sampling. 2019.

[48] A. Srinivasan, N. Nyapathy, B. Lee, S. M. Drucker, and J. Stasko. Collecting and characterizing natural language utterances for specifying data visualizations. In *CHI*, 2021.

[49] A. Srinivasan and J. Stasko. Orko: Facilitating multimodal interaction for visual exploration and analysis of networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):511–521, 2018.

[50] A. Srinivasan and J. T. Stasko. Natural language interfaces for data analysis with visualization: Considering what has and could be asked. In *EuroVis*, pp. 55–59. Eurographics Association, 2017.

[51] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 3104–3112, 2014.

[52] N. Tang, E. Wu, and G. Li. Towards democratizing relational data visualization. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019*. ACM, 2019.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, 2017.

[54] A. Wang, Y. Pruksachatkun, and et al. Superglue: A stickier benchmark for general-purpose language understanding systems. In *NIPS*, 2019.

[55] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461, 2018.

[56] R. V. Yampolskiy. Turing test as a defining feature of ai-completeness. In *Artificial Intelligence, Evolutionary Computing and Metaheuristics - In the Footsteps of Alan Turing*, vol. 427, pp. 3–17. Springer, 2013.

[57] B. Yu and C. T. Silva. Flowsense: A natural language interface for visual data exploration within a dataflow system. *IEEE TVCG*, pp. 1–11, 2020.

[58] T. Yu, R. Zhang, H. Er, S. Li, E. Xue, and et al. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In *EMNLP-IJCNLP*, pp. 1962–1979, 2019.